Resumo de Teoria dos Grafos (Prova 1)

→ DEFINIÇÕES

- Uma ARTICULAÇÃO é um vértice que, se removido, aumenta o núm. de componente conexas do grafo.
- Uma PONTE é uma aresta que, se removido, aumenta o núm. de componentes conexas do grafo.
- Um GRAFO SIMPLES é um grafo não dirijido, sem laços e existe no máximo uma aresta entre quaisquer dois vértices.
- Uma ÁRVORE é um grafo simples, acíclico e conexo.
- Cada um dos subgrafos conexos maximais de um grafo desconexo é chamado de COMPONENTE CONEXA.
- CAMINHO EULERIANO usa cada aresta somente uma vez. Um grafo deve possuir ∅ ou 2 vért. c/ grau ímpar p/ admitir um caminho euleriano. Para admitir um ciclo euleriano, todos os vért. devem possuir grau par.
- A soma dos graus de todos os vértices em um grafo não dirijido é sempre par e igual a 2 vezes o num. de arestas.
- Uma árvore de n vértices possui $n-1$ arestas.
- Um grafo simples de n vértices, possui um grau máximo p/ cada aresta de $n-1$.
- Um grafo completo de n vértices tem $n(n-1)/2$ arestas.
- Em um grafo, o número de vértices de grau ímpar é sempre par.
- Qualquer grafo conexo com n vértices possui no mínimo $n-1$ arestas.
-

→ IDENTIFICAÇÃO DE COMPONTES CONEXAS

```
int visit[N] ← -1
int ncc = 0;
for (int i = 0; i<N; i++){
    if (visit[i] == -1){
        ncc++;
        dfs(ncc, visit[N]); - - - - - >
    }
}
```

A única modificação é trocar visit[v] = 1 por visit[v] = ncc.

→ IDENTIFICAÇÃO DAS COMPONENTES BICONEXAS

1º) Gerar uma árvore por profundidade do grafo com as arestas de retorno

2º) Encontrar os Lowpt de cada vértice

3º) Identificar as articulações e demarcadores

4º) Identificar as comp. biconexas

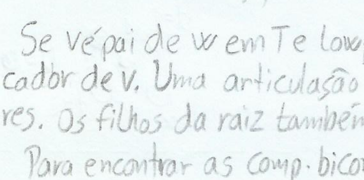
Um vértice v é uma articulasão se e somente se:
- V é raiz de T e possui mais de um filho
- V não é raiz de T, possui um filho w tal que lowpt(w) = v ou w

Se v é pai de w em T e lowpt(w) = v ou w, então w é demarcador de v. Uma articulasão é pai de um ou mais demarcadores. Os filhos da raiz também são demarcadores.

Para encontrar as comp. biconexas, escolhe-se um demarcador tal que sua subárvore não tenha articulações. Esta subárvore do demarcador, junto c/ o pai do demarcador, formam uma comp. biconexa. Retira-se a subárvore e, se o pai não possui mais demarcadores, não é mais articulasão.

→ ALGORITMO DE DIJKSTRA

A partir de um vértice inicial, a cada passo é selecionado, dentre os vértices remanescentes, o de menor valor. Durante o processo, mantem-se a dist. de cada vértice ao inicial. Inicialmente, o vert. inicial tem dist = 0 e os outros dist = ∞. Quando se seleciona um vért., atualiza-se a dist de seus adjacentes. Repete-se até chegar no vért. de dest.



→ GRAFOS BIPARTIDOS COMPLETOS

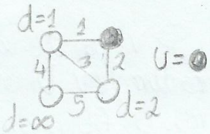
Denotados por K_{n_1,n_2}, onde $n_1 = |V_1|$, $n_2 = |V_2|$ e $n_1 \times n_2$ é o número de arestas.

→ DFS

```
int visit[N] = {0};
int G[N][N];
void dfs(int vertex){
    visit[vertex]= 1;
    for (int i = 0; i<N; i++){
        if(G[vertex][i] ==1 && visit[i]==0){
            dfs(i);
        }
    }
}
```

→ ALGORITMO DE PRIM

Escolhe um vértice inicial e marque como visitado. Selecione a aresta de menor peso que conecta um vért. do conjunto de visitados a um vértice não visitado. Adicione essa aresta à arvore e marque o vért. como visitado. Repita até que todos os vértices estejam incluídos.

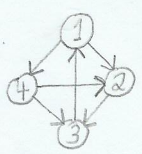


→ ALGORITMO DE KRUSKAL

Classifique todas as arestas em ordem crescente de peso. Itere sobre este conjunto e adicione à arvore as arestas que não formam ciclo c/ as arestas da árvore. Repita até que todos os vértices estejam incluídos.

→ ALGORITMO DE FLOYD-WARSHALL

Cria-se uma matriz 'dist' a partir do grafo dado. Se existe uma aresta entre os vértices i e j, dist[i][j]=w, caso contrário, dist[i][j] = ∞. Em dist[i][i] = 0. Agora, p/ cada par de vértices (i, j), verifica-se se existe um vért. K tal que o caminho $i \to K \to j$ é mais curto que $i \to j$. Se sim, atualizamos dist[i][j].



$$d^{(0)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \quad d^{(1)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix} \dots d^{(4)}$$

→ CONECTIVIDADE

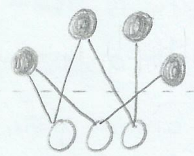
Se a partir de um vértice posso atingir todos os demais, é um grafo conexo. Se isso é verdade p/ qualq. vértice, o grafo é fortemente conexo. Se um grafo não dirijido é conexo, ele é fortemente conexo.

→ BFS

```
int nivel[N] = {-1};
void bfs(int iniVertex){
    int inicioFila = 0, fimFila = 1;
    int fila[N];
    fila[inicioFila]= iniVertex;
    nivel[iniVertex] = 0;
    while (inicioFila != fimFila){
        int verticeAtual = fila[inicioFila++];
        for (int i=0; i<N; i++){
            if(G[verticeAtual][i]==1 && nivel[i]==-1){
                fila[finalFila++] = i;
                nivel[i] = nivel[verticeAtual] +1;
            }
        }
    }
}
```

→ GRAFOS BIPARTIDOS

Posso identificar seus dois conjuntos c/ um algoritmo de colorasão que é uma modificasão do BFS. Uso um vetor de cores e vértices adjac. devem ter cores distintas.



→ ÁRVORE GERADORA MÍNIMA

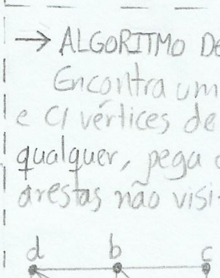
É uma árvore geradora que apresenta a menor soma de valores associados às arestas. Utiliza-se Prim ou Kruskal

→ PROBLEMA DO CAMINHO MÍNIMO

Se o grafo não é valorado, encontra-se c/ BFS.
Se o grafo é valorado, pode-se utilizar: Dijkstra ($w \in \mathbb{N}$), Bellman-Ford ($w \in \mathbb{Z}$), Floyd-Warshall (distância entre todos os pares de vértices), LCA (para árvores)

→ ALGORITMO DE HIERHOLZER

Encontra um ciclo euleriano a partir de um grafo conexo e c/ vértices de grau par. Seleciona um vértice e faz um ciclo qualquer, pega um vértice desse ciclo e cria um subciclo nas arestas não visitadas e junte os ciclos.



aebda
↓
bcfb = aebcfbda
↓
efde = aefdebcfbda